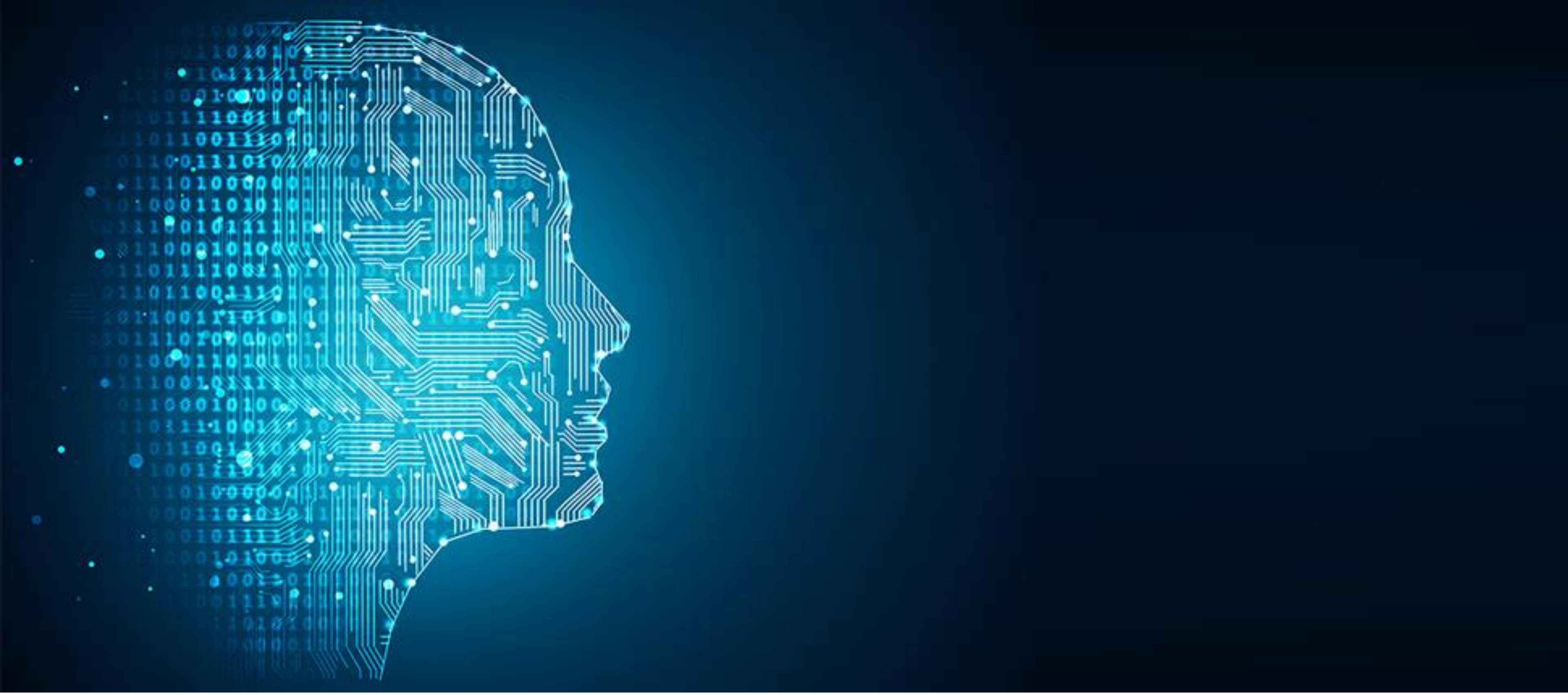


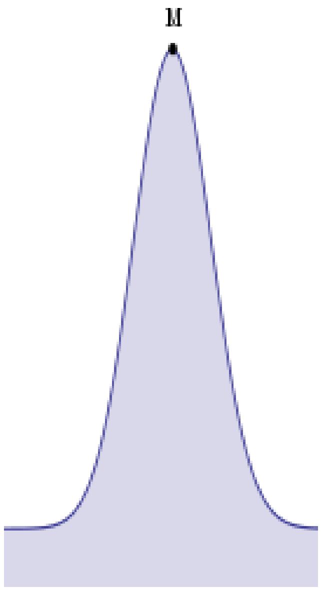
Artificial Intelligence Technologies

Search Techniques & Games II

By Hu Wang

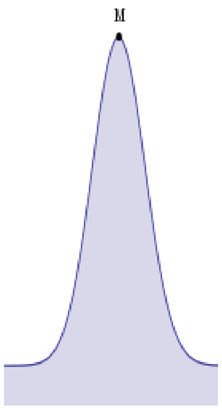
Search Techniques

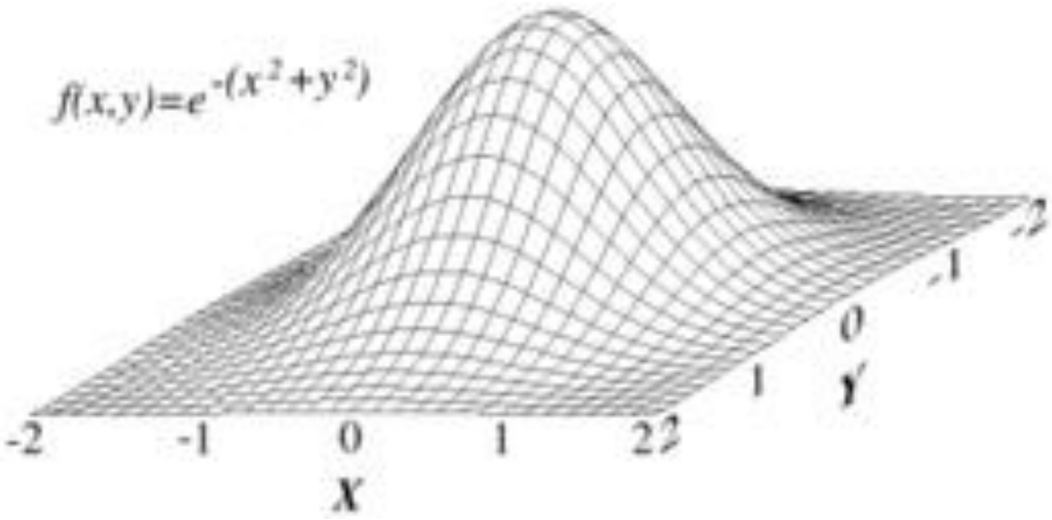
What is search techniques



How to find the maximum solution?
Optimization is commonly seen in our daily lives.

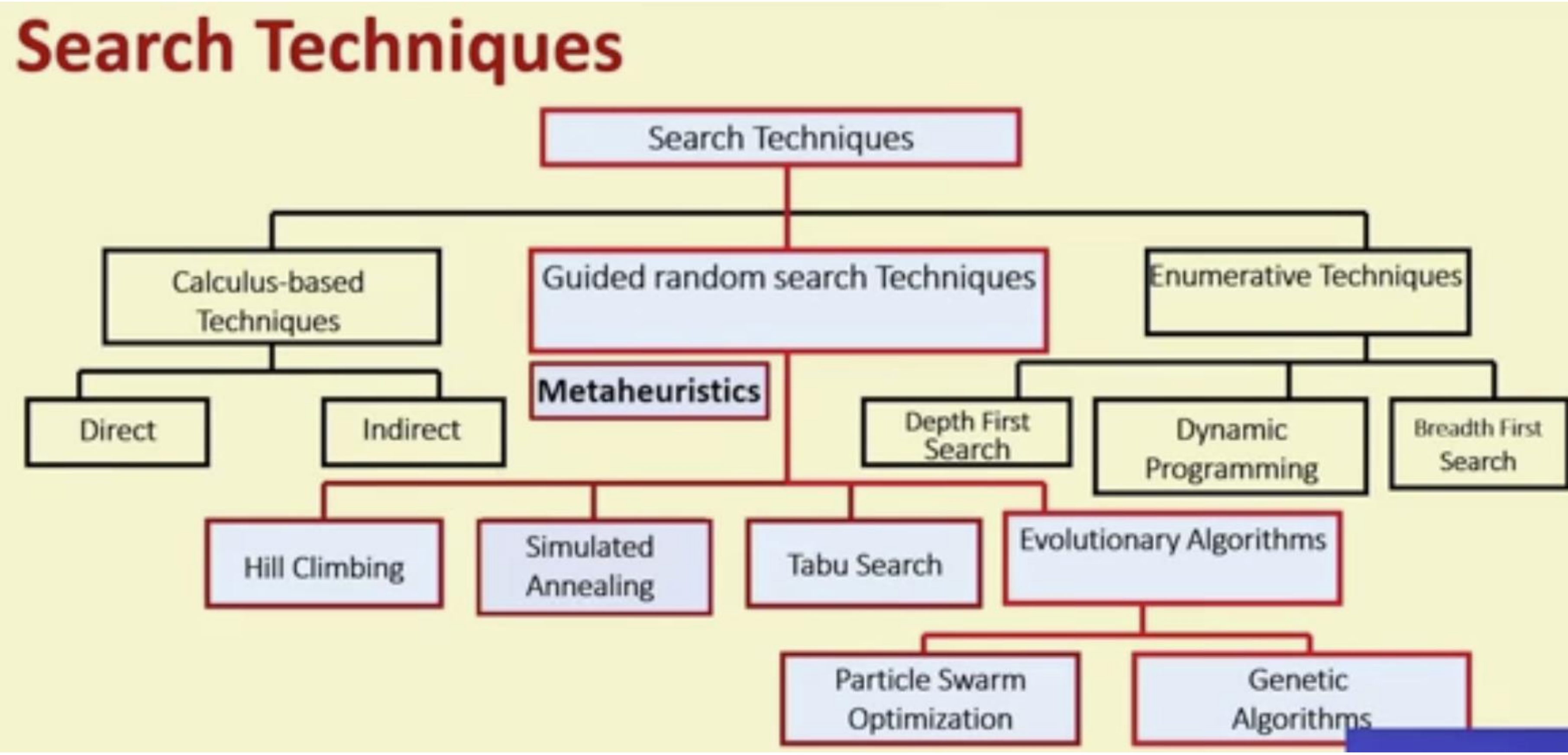
What is search techniques





How to find the maximum solution?
Optimization is commonly seen in our daily lives.

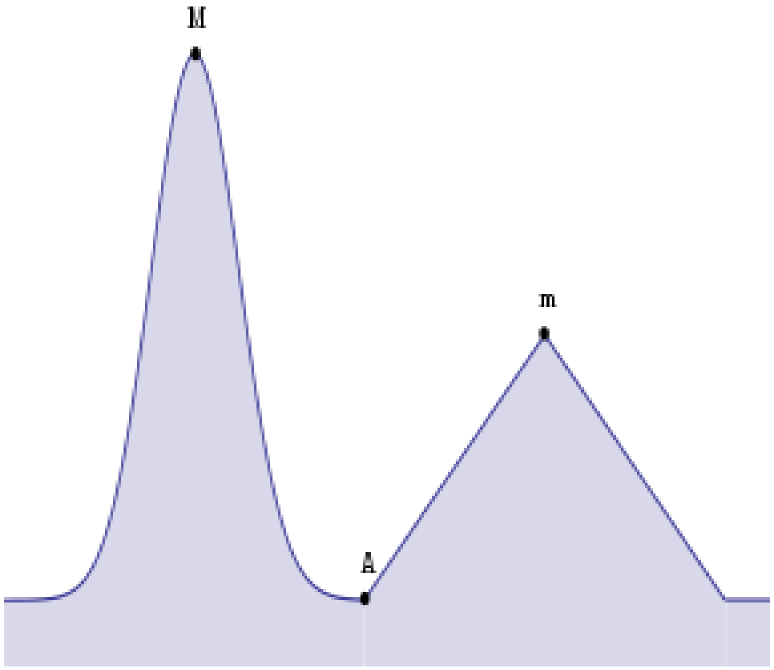
Search Techniques



Greedy Algorithm

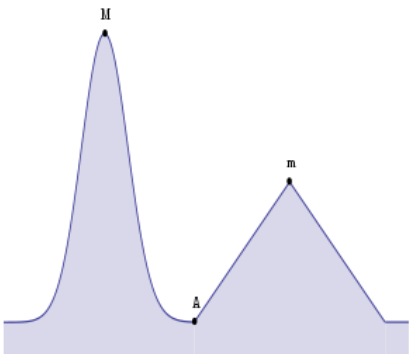
Greedy Algorithm

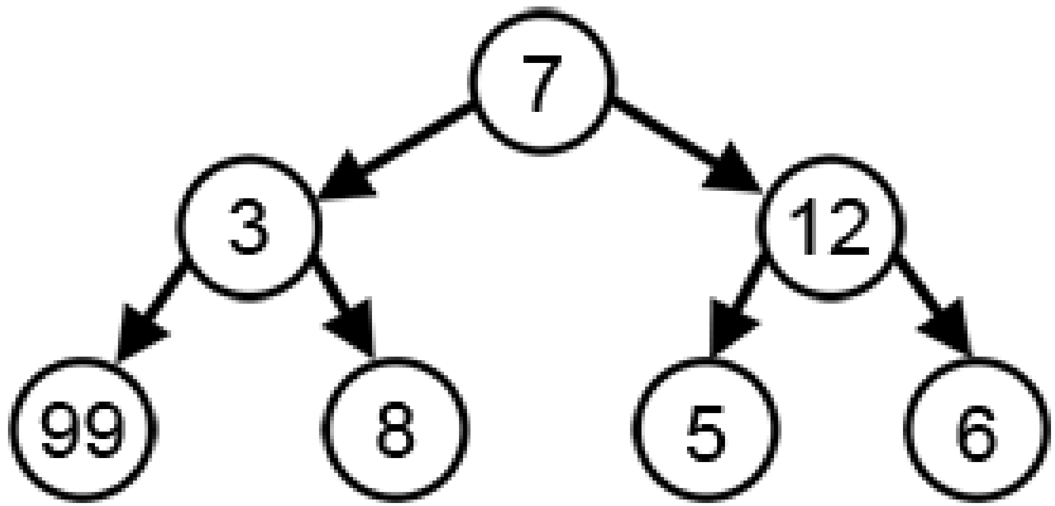
Greedy Algorithm



Starting from A, a greedy algorithm that tries to find the maximum by following the **greatest slope** will find the local maximum at "m", oblivious to the global maximum at "M".

Greedy Algorithm





With a goal of reaching the largest sum, at each step, the greedy algorithm will choose what appears to be the optimal immediate choice, so it will choose 12 instead of 3 at the second step, and will not reach the best solution, which contains 99.

Greedy Algorithm --- Steps

- Step 1: Start from an initial solution;
- Step 2: An iterative process: In a sub-task, take steps to **maximize** your own profits, and reduce the scale of the problem;
- Step 3: Combine all the sub-solutions.

Pseudo-code for Greedy Algorithm

```
Algorithm Greedy (a,n)
//a[1:n]contains the n inputs.
{
   solution:=0;//initialize the solution.
   for i:=1 to n do
   {
        x:=Select(a);
        if Feasible( solution, x) then
                solution:=Union(solution,x);
   }
   return solution;
}
```

Greedy Algorithm --- Have a think

- Suppose you have opened a small store and cannot pay electronically. The currency in the cash drawer is only **25 cents**, **10 cents**, **5 cents** and **1 cent**. If you are a salesperson and looking for coins of **41 cents** to customers, how can you arrange them to let the money for the customer is correct and the number of coins is the least?

Here are a few points that need to be clear:

- The currency has only four types of coins: 25 cents, 10 cents, 5 cents and 1 cent;
- Find correct coins of 41 cents for the customer;
- Minimize the number of coins

Greedy Algorithm --- Have a think

- Suppose you have opened a small store and cannot pay electronically. The currency in the cash drawer is only **25 cents**, **10 cents**, **5 cents** and **1 cent coins**. If you are a salesperson and looking for coins for **41 cents** to customers, how can you arrange them to let the money for the customer is correct and the number of coins is the least?

Could this problem be solved by Greedy Algorithms?

Greedy Algorithm --- Have a think

- Suppose you have opened a small store and cannot pay electronically. The currency in the cash drawer is only **25 cents**, **10 cents**, **5 cents** and **1 cent coins**. If you are a salesperson and looking for coins for **41 cents** to customers, how can you arrange them to let the money for the customer is correct and the number of coins is the least?

Could this problem be solved by Greedy Algorithms?

Greedy Algorithm --- Steps

- Step 1. If you can find 25 cent coins, don't use a 10 cent coin. For the first time, give customers 25 cent;
- Step 2. The customer have money=41-25=16 cents currently. Then choose the max one that can fit in 16, that is, 10 cents. Thus, 16-10=6 cents remaining. Repeat the process iteratively, until money=6-5=1, money=1-1=0. At this point, the customer receives all the changes and the transaction ends;
- Step 3. In the end, 41 points are divided into 1*25, 1*10, 1*5, 1*1 and a total of 4 coins.

Warning of Programming

Greedy Algorithm --- Code

```cpp
#include<iostream>
using namespace std;

#define ONECENT     1
#define FIVECENT     5
#define TENCENT     10
#define TWENTYFINECENT 25

int main()
{
    int cur_money=41;
    int num_25=0,num_10=0,num_5=0,num_1=0;

    //Try different denominations
    while(cur_money>=TWENTYFINECENT){ num_25++; cur_money -=TWENTYFINECENT; }

    while(cur_money>=TENCENT){ num_10++; cur_money -=TENCENT; }

    while(cur_money>=FIVECENT){ num_5++; cur_money -=FIVECENT; }

    while(cur_money>=ONECENT){ num_1++; cur_money -=ONECENT; }

    //output
    cout<< "25 cents: "<<num_25<<endl;
    cout<< "10 cents: "<<num_10<<endl;
    cout<< "5 cents: "<<num_5<<endl;
    cout<< "1 cents: "<<num_1<<endl;

    return 0;
}
```

```cpp
#define ONECENT    1
#define FIVECENT    5
#define TENCENT    10
#define TWENTYFINECENT 25

int main()
{
    int cur_money=41;
    int num_25=0,num_10=0,num_5=0,num_1=0;

    //Try different denominations
    while(cur_money>=TWENTYFINECENT){ num_25++; cur_money -=TWENTYFINECENT; }

    while(cur_money>=TENCENT){ num_10++; cur_money -=TENCENT; }

    while(cur_money>=FIVECENT){ num_5++; cur_money -=FIVECENT; }

    while(cur_money>=ONECENT){ num_1++; cur_money -=ONECENT; }

    //output
    cout<< "25 cents: "<<num_25<<endl;
    cout<< "10 cents: "<<num_10<<endl;
    cout<< "5 cents: "<<num_5<<endl;
    cout<< "1 cents: "<<num_1<<endl;

    return 0;
}
```

```
25 cents: 1
10 cents: 1
5 cents: 1
1 cents: 1
```

Demo

Hill Climbing Algorithm

Hill Climbing Algorithm

Hill Climbing Algorithm

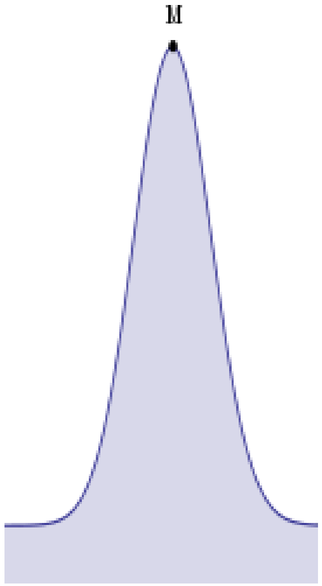
Hill Climbing Algorithm --- An example

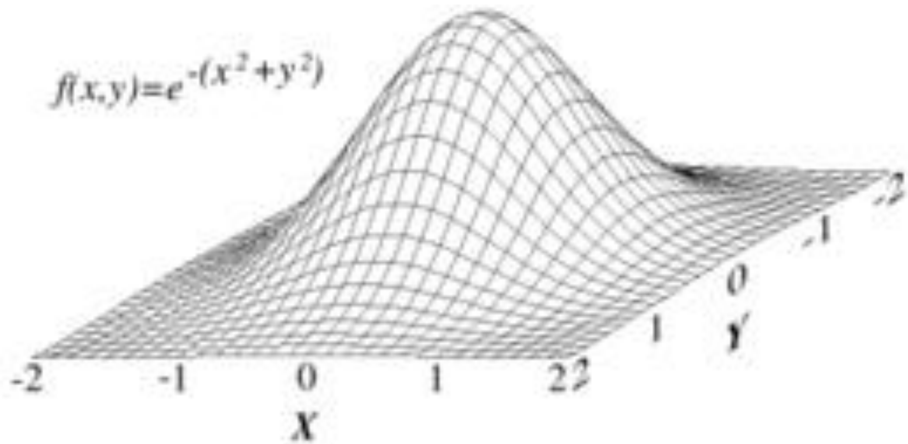
Find the maximum value between [5,8].

Demo

Hill Climbing Algorithm

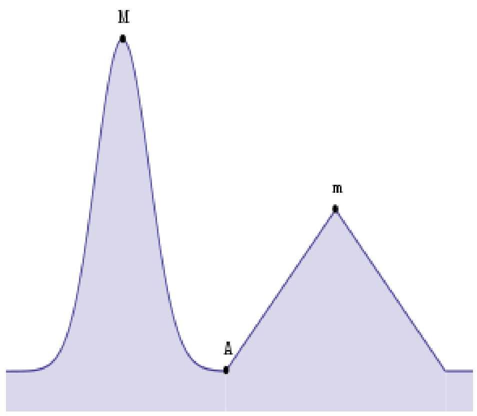
Hill Climbing Algorithm is a greedy algorithm.

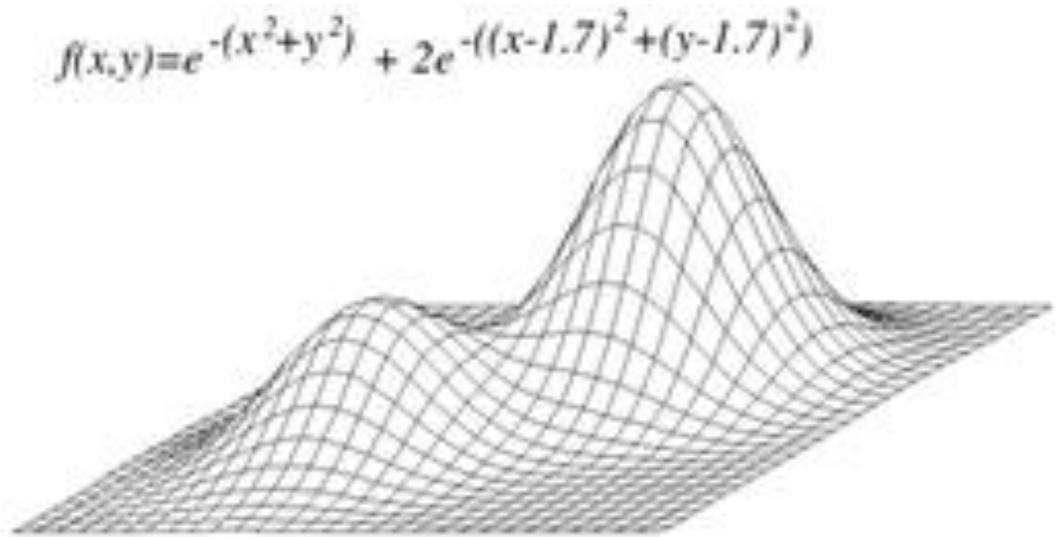




*A surface with only **one** maximum. Hill-climbers are well-suited for optimizing over such surfaces, and will converge to the global maximum.*

Hill Climbing Algorithm --- Problem

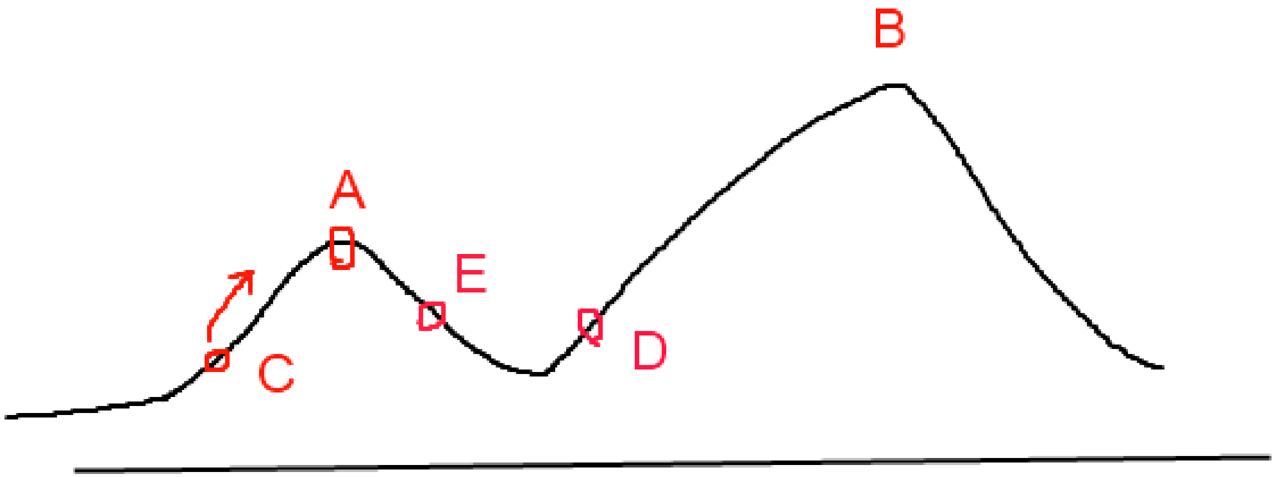




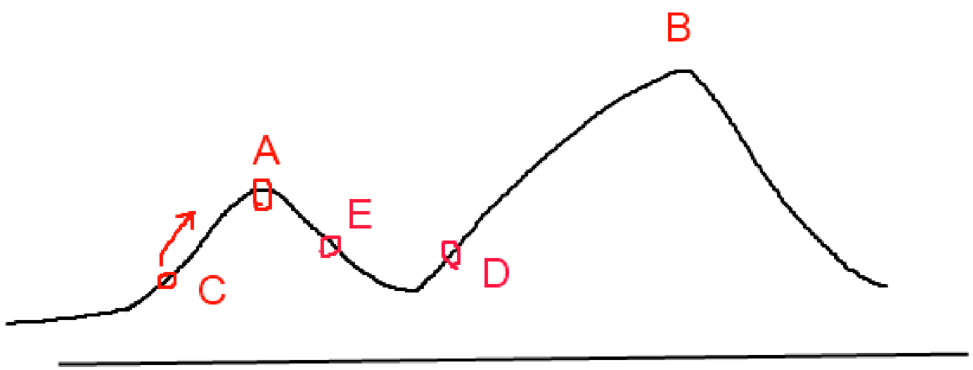
*A surface with **two** local maxima. (Only one of them is the global maximum.) If a hill-climber begins in a poor location, it may converge to the lower maximum.*

Simulated Annealing Algorithm

Simulated Annealing Algorithm VS Hill Climbing



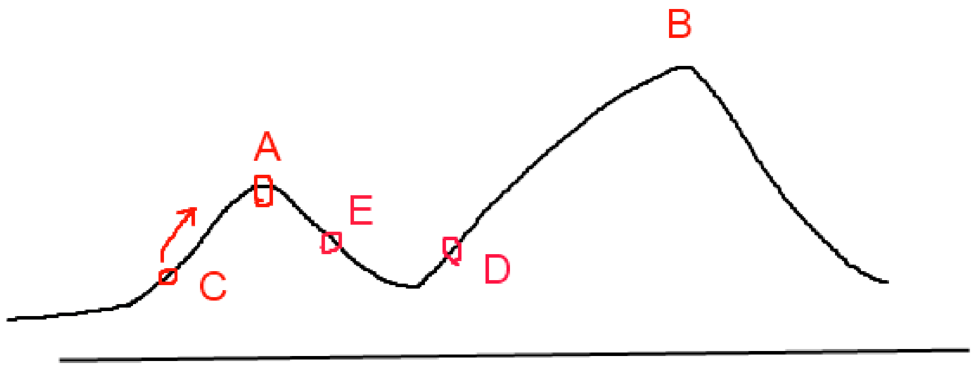
Simulated Annealing Algorithm VS Hill Climbing



The hill-climbing method is a **completely greedy** method. Every time it chooses a current optimal solution which is short-sighted.

Simulated annealing is a **greedy algorithm** as well, but its search process introduces random factors. The simulated annealing algorithm allows model accept a solution that is **worse** than the current solution with a **certain probability**, so it may jump out of this local optimal solution and reach the global optimal solution.

Simulated Annealing Algorithm VS Hill Climbing



Taking the Figure as an example, the simulated annealing algorithm will accept the movement of E with a certain probability after searching for the local optimal solution A. Maybe after a few such moves, point D will be reached, so the local maximum A will be jumped out.

Simulated Annealing Algorithm --- Temperature

$$P = \begin{cases} 1 & E(x_{\text{new}}) > E(x_{\text{old}}) \\ \exp\left(-\frac{E(x_{\text{old}}) - E(x_{\text{new}})}{T}\right), & E(x_{\text{new}}) \leq E(x_{\text{old}}) \end{cases}$$

1. Get a better solution after moving. Then always accept the move
2. That is, the solution after the move is worse than the current solution, then the move is accepted with a certain probability, and this probability is gradually reduced over time (gradually reduced to stabilize)

The calculation of "a certain probability" here refers to the annealing process of metal smelting, which is the origin of the name of the simulated annealing algorithm. It is calculated according to the principles of thermodynamics.

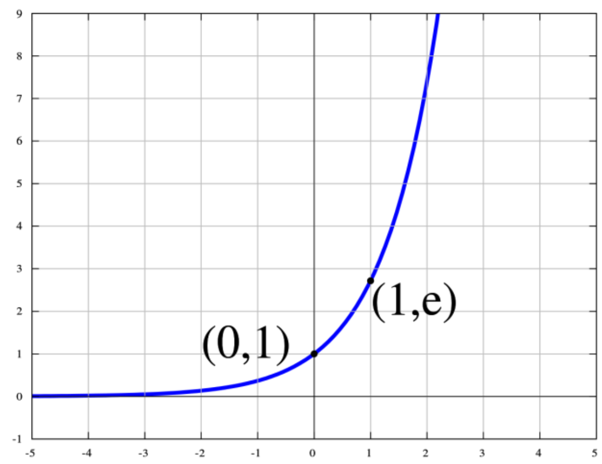
Simulated Annealing Algorithm --- Temperature

It is related to the current temperature parameter T, which decreases with the decrease of temperature.

$$P = \begin{cases} 1 & E(x_{\text{new}}) > E(x_{\text{old}}) \\ \exp\left(-\frac{E(x_{\text{old}}) - E(x_{\text{new}})}{T}\right), & E(x_{\text{new}}) \le E(x_{\text{old}}) \end{cases}$$

The classic simulated annealing algorithm:

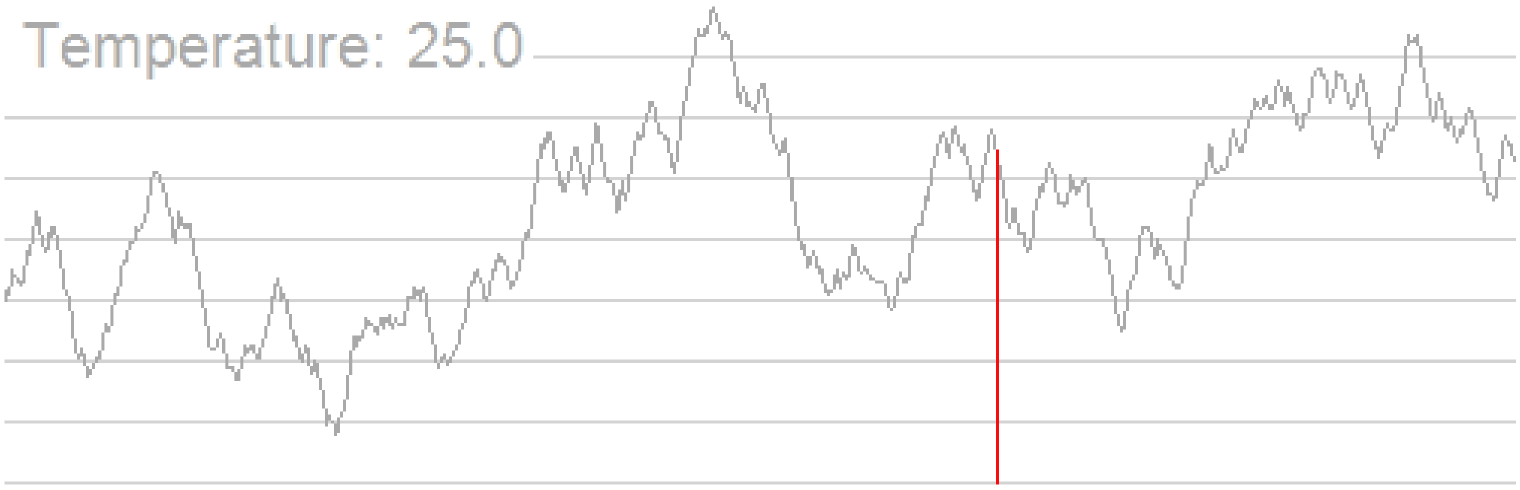
$$T(t) = \frac{T_0}{\lg(1+t)}$$



The fast simulated annealing algorithm:

$$T(t) = \frac{T_0}{1+t}$$

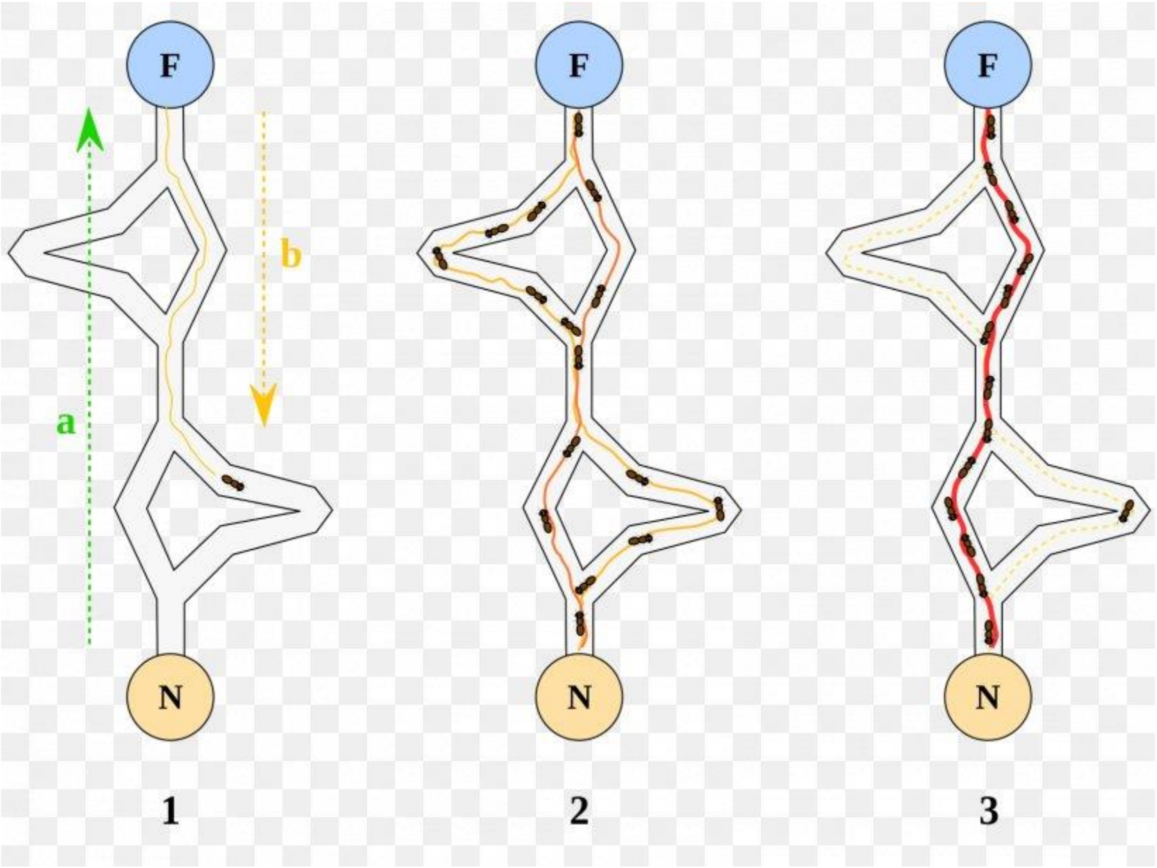
Simulated Annealing Algorithm



Despite the many local maxima in this graph, the global maximum can still be found using **simulated annealing**. Unfortunately, the applicability of simulated annealing is problem-specific because it relies on finding *lucky jumps* that improve the position. In such extreme examples, hill climbing will most probably produce a local maximum.

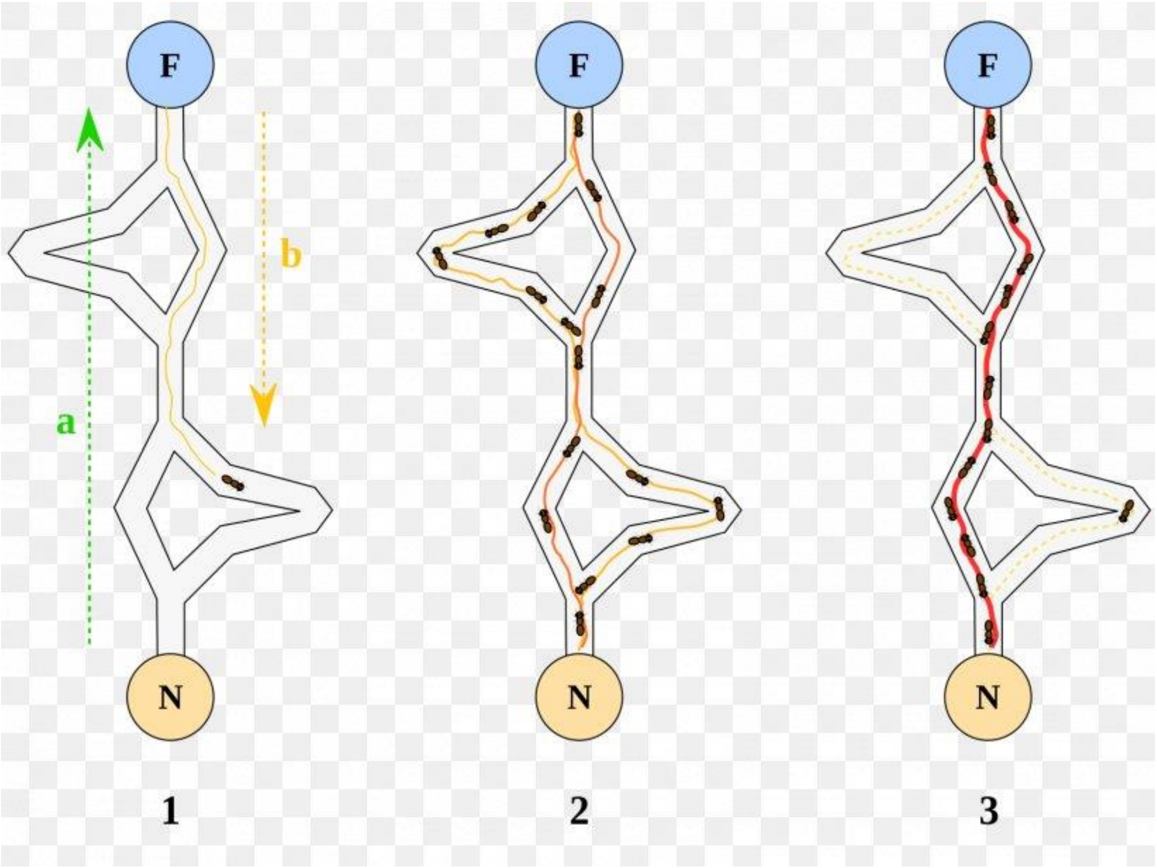
Ant Colony Optimization Algorithm

Ant Colony Optimization Algorithm



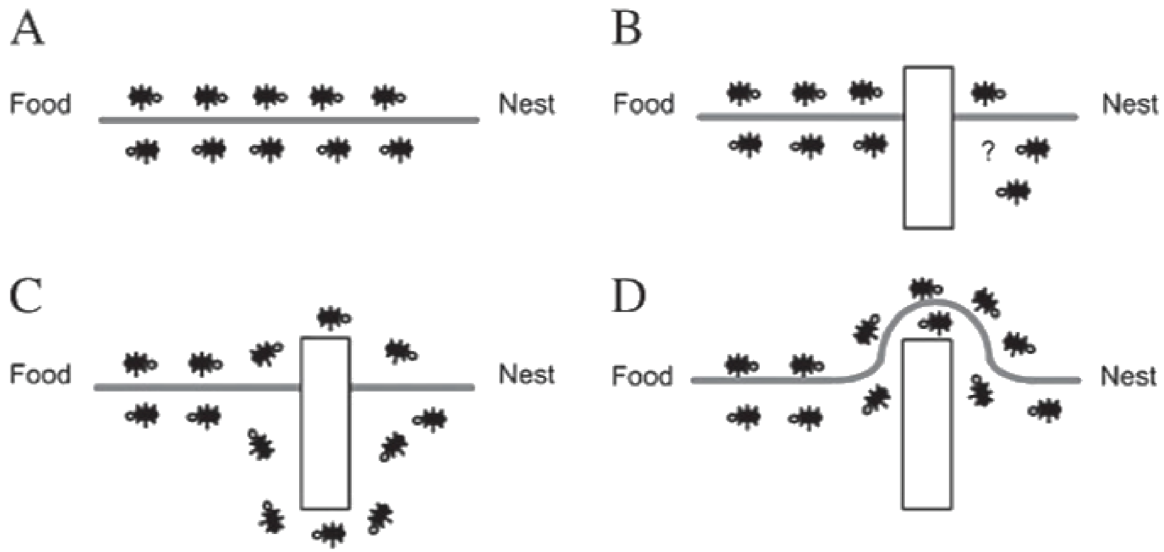
Individual ant acts with randomness;
*but there are **patterns in colony actions***

Ant Colony Optimization Algorithm



Invented by Italian scientist Marco Dorigo in 1992

Ant Colony Optimization Algorithm

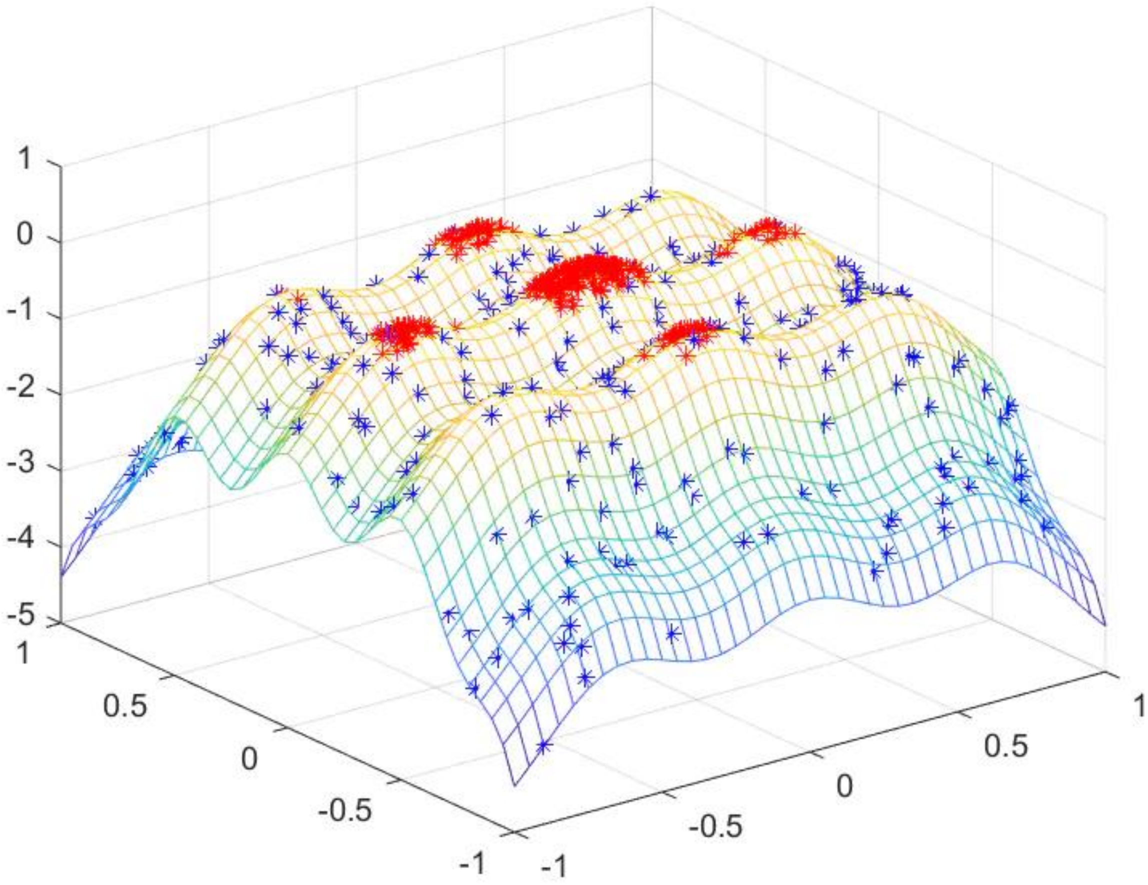


Pheromone Tracking: following routes with strong pheromone within a certain probability

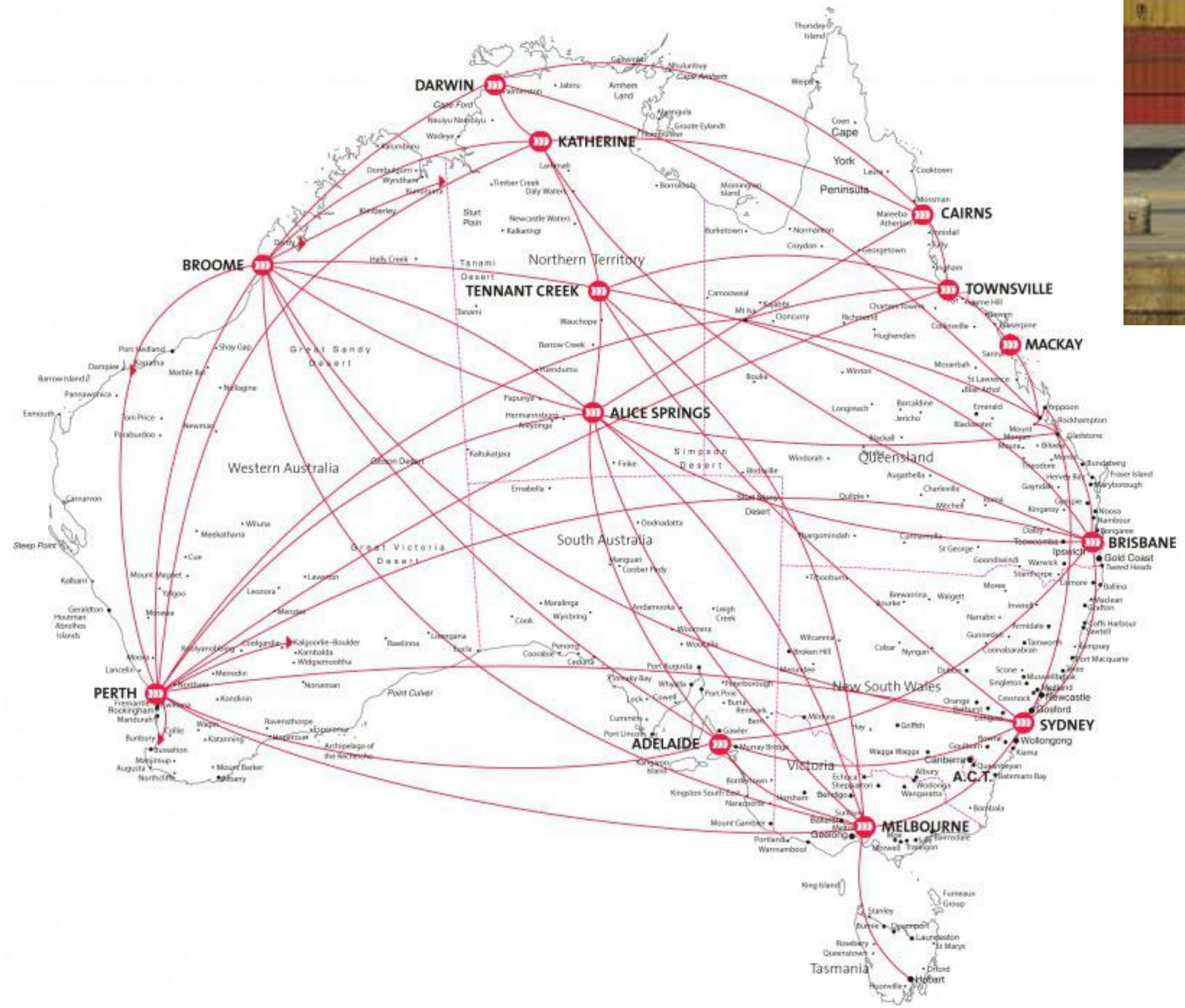
Pheromone residue: leave pheromone behind. These pheromone will fade away with time. The closer an ant to the food, the more Pheromone will be released.

Ant Colony Optimization Algorithm

Ant Colony Optimization Algorithm



Ant Colony Optimization Algorithm



Reference

- Greedy algorithm

  https://en.wikipedia.org/wiki/Greedy_algorithm

- Hill climbing & Simulated annealing

  https://en.wikipedia.org/wiki/Hill_climbing

- Ant colony optimization algorithms

  https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms